CLAIMS

What is claimed is:

1. A method for implementing transaction services patterns, comprising the steps of:

(a) batching logical requests for reducing network traffic;

(b) allowing a batched request to indicate that it depends on the response to another request;

(c) sending a single message to all objects in a logical unit of work;

(d) sorting requests that are being unbatched from a batched message; and

(e) assigning independent copies of business data to concurrent logical units of work for helping prevent the logical units of work from interfering with each other.

2. A method as recited in claim 1, wherein the step of batching logical requests includes the steps of: providing a group of business objects necessary for a transaction; managing the group of business objects necessary to the transaction in a logical unit of work; grouping logically-related requests received from the logical unit of work into a single network message; storing the message; and sending the message upon receiving an order to send the message.

3. A method as recited in claim 1, wherein the step of allowing a batched request to indicate that it depends on the response to another request includes the steps of: providing a group of business objects necessary for a transaction; batching logically-related requests received from the business objects into a single network message, wherein one of the requests is a parent request; receiving a register that at least one of the requests is dependent upon the response data from the parent request; sending the network message across a network; unbundling the requests from the network message; receiving a response to the parent request; directing data from the response to

from?

the parent request to the dependent request; and receiving a response to the dependent request based on the received data from the response to the parent request.

4. A method as recited in claim 1, wherein the step of sending a single message to all objects includes the steps of: providing a group of business objects necessary for a transaction; managing the group of business objects necessary to the transaction in a logical unit of work; creating a receiver which communicates with the business objects in the logical unit of work; receiving a message for the business objects in the logical unit of work; and directing the message to the receiver, wherein the receiver forwards the message to each of the business objects in the logical unit of work.

5. A method as recited in claim 1, wherein the step of sorting requests that are being unbatched from a batched message includes the steps of: providing a group of business objects necessary for a transaction; grouping logically-related requests received from the business objects; obtaining at least one of sorting rules and sort weights; sorting the requests in the message and placing them in a specific order determined from the one of the sorting rules and the sort weights; batching the sorted requests into a single message; sending the message to a data server; and unbundling the requests from the message in the specific order.

6. A method as recited in claim 1, wherein the step of assigning independent copies of business data to concurrent logical units of work for helping prevent the logical units of work from interfering with each other includes the steps of: providing multiple logical units of work operating concurrently, wherein each of the logical units of work manipulate at least one common business object; creating a copy of the common business object for each of the logical units of work such that the copy of the business object for one logical unit of work becomes a separate instance from the copy of the

business object for another logical unit of work, wherein each copy of the business object knows the context of that copy of the business object in relation to the associated logical unit of work; receiving a request to make changes to a copy of the business object of one of the logical units of work and changing that copy of the business object, wherein the other copies of the business object are not changed; verifying that only one copy of the business object has been changed; and updating the common business object based on the change to the copy of the business object.

7. A computer program embodied on a computer readable medium for implementing transaction services patterns, comprising:

(a) a code segment that batches logical requests for reducing network traffic;

(b) a code segment that allows a batched request to indicate that it depends on the response to another request;

(c) a code segment that sends a single message to all objects in a logical unit of work;

(d) a code segment that sorts requests that are being unbatched from a batched message; and

(e) a code segment that assigns independent copies of business data to concurrent logical units of work for helping prevent the logical units of work from interfering with each other.

8. A computer program as recited in claim 7, wherein the code segment that batches logical requests includes: a code segment that provides a group of business objects necessary for a transaction; a code segment that manages the group of business objects necessary to the transaction in a logical unit of work; a code segment that groups logically-related requests received from the logical unit of work into a single network message; a code segment that stores the message; and a code segment that sends the message upon receiving an order to send the message.

9. A computer program as recited in claim 7, wherein the code segment that allows a batched request to indicate that it depends on the response to another request includes: a code segment that provides a group of business objects necessary for a transaction; a code segment that batches logically-related requests received from the business objects into a single network message, wherein one of the requests is a parent request; a code segment that receives a register that at least one of the requests is dependent upon the response data from the parent request; a code segment that sends the network message across a network; a code segment that unbundles the requests from the network message; a code segment that receives a response to the parent request; a code segment that directs data from the response to the parent request to the dependent request; and a code segment that receives a response to the dependent request based on the received data from the response to the parent request.

10. A computer program as recited in claim 7, wherein the code segment that sends a single message to all objects includes: a code segment that provides a group of business objects necessary for a transaction; a code segment that manages the group of business objects necessary to the transaction in a logical unit of work; a code segment that creates a receiver which communicates with the business objects in the logical unit of work; a code segment that receives a message for the business objects in the logical unit of work; and a code segment that directs the message to the receiver, wherein the receiver forwards the message to each of the business objects in the logical unit of work.

11. A computer program as recited in claim 7, wherein the code segment that sorts requests that are being unbatched from a batched message includes: a code segment that provides a group of business objects necessary for a transaction; a code segment that groups logically-related requests received from the business objects; a code segment that obtains at least one of sorting

rules and sort weights; a code segment that sorts the requests in the message and placing them in a specific order determined from the one of the sorting rules and the sort weights; a code segment that batches the sorted requests into a single message; a code segment that sends the message to a data server; and a code segment that unbundles the requests from the message in the specific order.

12. A computer program as recited in claim 7, wherein the code segment that assigns independent copies of business data to concurrent logical units of work for helping prevent the logical units of work from interfering with each other includes: a code segment that provides multiple logical units of work operating concurrently, wherein each of the logical units of work manipulate at least one common business object; a code segment that creates a copy of the common business object for each of the logical units of work such that the copy of the business object for one logical unit of work becomes a separate instance from the copy of the business object for another logical unit of work, wherein each copy of the business object knows the context of that copy of the business object in relation to the associated logical unit of work; a code segment that receives a request to make changes to a copy of the business object of one of the logical units of work and changing that copy of the business object, wherein the other copies of the business object are not changed; a code segment that verifies that only one copy of the business object has been changed; and a code segment that updates the common business object based on the change to the copy of the business object.

13. A system for implementing transaction services patterns, comprising:

(a) logic that batches logical requests for reducing network traffic;

(b) logic that allows a batched request to indicate that it depends on the response to another request;

(c) logic that sends a single message to all objects in a logical unit of work;

(d) logic that sorts requests that are being unbatched from a batched message; and

(e) logic that assigns independent copies of business data to concurrent logical units of work for helping prevent the logical units of work from interfering with each other.

14. A system as recited in claim 13, wherein the logic that batches logical requests includes: logic that provides a group of business objects necessary for a transaction; logic that manages the group of business objects necessary to the transaction in a logical unit of work; logic that groups logically-related requests received from the logical unit of work into a single network message; logic that stores the message; and logic that sends the message upon receiving an order to send the message.

15. A system as recited in claim 13, wherein the logic that allows a batched request to indicate that it depends on the response to another request includes: logic that provides a group of business objects necessary for a transaction; logic that batches logically-related requests received from the business objects into a single network message, wherein one of the requests is a parent request; logic that receives a register that at least one of the requests is dependent upon the response data from the parent request; logic that sends the network message across a network; logic that unbundles the requests from the network message; logic that receives a response to the parent request; logic that directs data from the response to the parent request to the dependent request; and logic that receives a response to the dependent request based on the received data from the response to the parent request.

16. A system as recited in claim 13, wherein the logic that sends a single message to all objects includes: logic that provides a group of business objects necessary for a transaction; logic that manages the group of business objects necessary to the transaction in a logical unit of work; logic that

creates a receiver which communicates with the business objects in the logical unit of work; logic that receives a message for the business objects in the logical unit of work; and logic that directs the message to the receiver, wherein the receiver forwards the message to each of the business objects in the logical unit of work.

17. A system as recited in claim 13, wherein the logic that sorts requests that are being unbatched from a batched message includes: logic that provides a group of business objects necessary for a transaction; logic that groups logically-related requests received from the business objects; logic that obtains at least one of sorting rules and sort weights; logic that sorts the requests in the message and placing them in a specific order determined from the one of the sorting rules and the sort weights; logic that batches the sorted requests into a single message; logic that sends the message to a data server; and logic that unbundles the requests from the message in the specific order.

18. A system as recited in claim 13, wherein the logic that assigns independent copies of business data to concurrent logical units of work for helping prevent the logical units of work from interfering with each other includes: logic that provides multiple logical units of work operating concurrently, wherein each of the logical units of work manipulate at least one common business object; logic that creates a copy of the common business object for each of the logical units of work such that the copy of the business object for one logical unit of work becomes a separate instance from the copy of the business object for another logical unit of work, wherein each copy of the business object knows the context of that copy of the business object in relation to the associated logical unit of work; logic that receives a request to make changes to a copy of the business object of one of the logical units of work and changing that copy of the business object, wherein the other copies of the business object are not changed; logic that verifies that only one copy

of the business object has been changed; and logic that updates the common business object based on the change to the copy of the business object.